Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Novak Hindel

Obdelava videa v realnem času v orodju Orange

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Janez Demšar

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Orange je aplikacija za podatkovno rudarjenje, ki uporabniku omogoča programiranje postopka analize tako, da sestavlja sheme, po katerih se pretakajo podatki. Zaradi svoje fleksibilnosti in hkrati preprostosti je Orange zanimiv tudi za druga področja. Eden od zanimivih primerov je procesiranje videa, ki predstavlja dodaten izziv zato, ker se mora odvijati v realnem času.

Razvijte zbirko gradnikov za procesiranje videa, ki bodo brali video iz datoteke, ga obdelovali in prikazali. Pri testiranju razvitih gradnikov bodite posebej pozorni na sinhronizacijo delov sheme z različnimi hitrostmi procesiranja.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Marko Novak Hindel sem avtor diplomskega dela z naslovom:

Vzorec diplomskega dela

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Janeza Demšarja
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne Podpis avtorja:

Zahvaljujem se svojim bližnjim za potrpljenje in podporo. Posebno se zahvaljujem svojemu mentorju, izr. prof. dr. Janezu Demšarju, za nasvete, vodstvo in pomoč pri izdelavi diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
BGR	blue, green, red	modra, zelena, rdeča
RGB	red, green, blue	rdeča, zelena, modra
fps	frame per second	okvirji na sekundo
HSV	hue, saturation, value	odtenek, zasičenost, svetlost

Povzetek

Cilj diplomske naloge je razvoj in predstavitev skupka orodij za urejanje videa, razvitih za orodje za podatkovno rudarjenje Orange. V prvem delu naloge je predstavljena motivacija za delo. V glavnem delu najprej predstavimo orodja in tehnologije, ki smo jih uporabili pri razvoju, nato sledi predstavitev zgradbe razvitih gradnikov in predstavitev razvitih grafičnih gradnikov. V zadnjem delu predstavimo zaključek, ter možnosti za nadaljnje delo in izboljšave.

Ključne besede: Obdelava videa, Python, Orange, OpenCV, Numpy.

Abstract

The aim of this thesis is to present the development of a set of video processing tools for the Orange data mining toolkit. In the first part of this thesis we present the motivation for our work. We then describe the tools and libraries used in development. After that we describe the structure of the developed widgets and present the widgets developed for this thesis. In conclusion we sumarize our work and present the options for further development and improvement.

Keywords: Video Processing, Python, Orange, OpenCV, Numpy.

Poglavje 1

Uvod

Na različnih področjih znanosti pogosto delamo z videom v realnem času. Zajete posnetke je pogosto potrebno ustrezno filtrirati, če želimo iz njih izvleči želene podatke. Za obdelavo videa obstaja mnogo različnih programskih rešitev, a je zelo koristno, če je mogoče že uporabljeno aplikacijo razširiti v skladu z novimi potrebami. V tem je tudi bistvo naloge, ki jo predstavljamo. Razširiti orodje za podatkovno rudarjenje Orange z možnostjo obdelave videa v realnem času.

V okviru naloge smo razvili 13 grafičnih gradnikov s katerimi želimo predstaviti možnost urejanja videa v realnem času znotraj orodja Orange. Grafični gradniki omogočajo branje video datotek in predvajanje le-teh v realnem času. Poleg tega imamo še 10 grafičnih gradnikov, s katerimi lahko izvajamo nekatere pogoste obdelave videa kot so zrcaljenje, ravnanje histograma, upragovanje...

V naslednjih poglavjih so najprej predstavljena orodja in tehnologije s katerimi smo delali. Za tem sledi predstavitev zgradbe razvitih grafičnih gradnikov. Na koncu predstavimo uporabo in implementacijo vseh trinajstih grafičnih gradnikov, razvitih v okviru naloge.

Poglavje 2

Uporabljena orodja in tehnologije

Orodje Orange[1] je implementirano v programskem jeziku Python[2], v okviru naloge smo razvijali za Orange 3, ki je implementiran v programskem jeziku Python 3. Orange uporablja še knjižnice Numpy[3] (podatkovne strukture za hranjenje podatkov), ter PyQt[4] (za uporabniški vmesnik). Za branje video datotek in izvajanje nekaterih operacij obdelave videa je bila uporabljena knjižnica OpenCV[5].

2.1 Python

Python je interpretiran visoko-nivojski objektno orientiran programski jezik dinamičnih tipov. Programski jezik je razvil Guido van Rossum v zgodnjih devetdesetih letih z namenom da je enostaven za branje, a da vseeno omogoča dostop do nizko-nivojskih možnosti, ter da omogoča lahko prenosljivost [6].

[1] http://orange.biolab.si
[2] https://www.python.org
[3] http://www.numpy.org
[4] https://www.riverbankcomputing.com/software/pyqt/intro
[5] http://opencv.org

Python nam med drugim ponuja:

- preprosto sintakso, s katero lahko pišemo proceduralno ali popolnoma objektno orientirano kodo,
- zmogljiv interpreter,
- obsežno zbirko modulov za različna področja,
- možnost uporabe programskega jezika znotraj že razvitih aplikacij,
- enoten repozitorij modulov,

ter vrsto ostalih prednosti [5].

Python vsebuje tudi vmesnik do programskega jezika C, kar nam omogoča optimizacijo najbolj računsko zahtevnih delov aplikacij [6].

Zaradi preproste sintakse se je Python uveljavil kot programski jezik za prototipiranje algoritmov. Z razvojem paketov kot sta Numpy in SciPy pa je postal zelo priljubljen v znanstvenem računalništvu [6][5].

Kot pomanjkljivost programskega jezika Python lahko omenimo sočasni obstoj dveh različnih verzij: Python 2 (trenutna verzija 2.7.10) in Python 3 (trenutna verzija 3.5.0). Verzija 3 je uvedla večjo konsistentnost jezika in predstavlja prihodnost jezika. Verzija 2 se je obdržala v uporabi zaradi že obstoječih modulov, katerih selitev na verzijo 3 je potekala počasi [4].

Vsa programska koda, ki smo jo razvili v okviru te naloge je napisana v Pythonu 3.

2.2 Numpy

Zahvaljujoč paketu Numpy je Python postal priljubljen pri znanstvenikih in inženirjih. V osnovi prispeva *array*, homogeno, več-dimenzionalno polje določenega podatkovnega tipa. Polje vsebuje množico atributov in metod, s pomočjo katerih lahko učinkovito delamo z vsebovanimi podatki [5].

Poleg objekta *array* nam Numpy ponuja tudi množico univerzalnih funkcij, s katerimi lahko izvajamo operacijo nad n vhodi, element za elementom.

Osnovne matematične operacije, ki se izvajajo nad polji, so implementirane s pomočjo univerzalnih funkcij. Poleg že obstoječih univerzalnih funkcij imamo tudi možnost definiranja lastnih funkcij [5].

Razen osnovnega razreda *array* in univerzalnih funkcij imamo na voljo še vrsto funkcij za delo s polji in izvajanje različnih matematičnih operacij [5].

Python sicer že vsebuje bogato zbirko podatkovnih struktur (seznami, slovarji...), vendar so le-ti neučinkoviti za numerično računanje [10]. Numpy programski jezik Python približa specializiranim jezikom za znanstveno računalništvo, kot je na primer Matlab.

Numpy polja uporablja veliko drugih paketov. So osnovna podatkovna struktura v Orange 3 in v paketu OpenCV.

V nalogi se Numpy polje uporablja za shranjevanje okvirjev videa. Okvir je predstavljen kot polje dimenzij $visina \times sirina \times 3$. Tako lahko na primer osamimo posamezen barvni kanal okvirja tako, da ostalima dvema preprosto nastavimo vrednosti na 0, kot je prikazano v kodi 2.1.

Koda 2.1: Primer izoliranja barvnega kanala

```
frame_red[:, :, 1] = 0
frame_red[:, :, 2] = 0
```

2.3 Orange

Orange je zbirka orodij za strojno učenje in podatkovno rudarjenje, namenjena uporabi preko Python skript ali vizualnega programiranja [2]. Sprva je bil Orange zamišljen kot C++ knjižnica algoritmov za strojno učenje. Ker je bilo to preveč omejujoče, je bil napisan vmesnik za Python [3]. Število modulov napisanih v Pythonu je kmalu preseglo število tistih napisanih v jeziku C++, uporaba Pythona je tudi poenostavila razvoj grafičnega uporabniškega vmesnika [3].

Knjižnica je oblikovana kot hierarhično organizirana zbirka orodij za po-

datkovno rudarjenje. Kompleksni algoritmi na višjih nivojih so sestavljeni iz enostavnejših procedur nižjih nivojev. To razvijalcem omogoča enostaven razvoj novih funkcionalnosti na podlagi že razvitih. Knjižnico je mogoče uporabljati v Python skriptah, poleg tega predstavlja temelj orodja za vizualno programiranje [2].

Z Orange verzijo 3 je bil C++ v celoti opuščen in zamenjan z modulom Scikit-learn[6], ki vsebuje orodja za strojno učenje.

V orodju za vizualno programiranje so komponente predstavljene z grafičnimi gradniki (ang. Widget). Te gradnike povezujemo med sabo, povezave predstavljajo podatke, ki se posredujejo od gradnika do gradnika.

Razvoj grafičnih gradnikov za Orange je zahvaljujoč modularni zasnovi knjižnice dokaj preprost. Vsak grafični gradnik je svoj razred, ki deduje razred *OWWidget*.

Razred *OWWidget* predstavlja Orangevo ovojnico PyQtja. Razred je osnova za razvoj grafičnih gradnikov za Orange. Vsebuje parametre, ki opisujejo gradnik znotraj grafičnega vmesnika: ime, id, verzija, kategorija, opis, pot do grafične ikone, avtor, ključne besede...

Ko razvijamo nov grafični gradnik za Orange definiramo te parametre, poleg tega lahko definiramo še množico vhodov in izhodov. Za vsak vhod moramo definirati ime vhoda, podatkovni tip, ki ga vhod prejme in ime metode, ki se kliče ob prejetju. Za vsak izhod moramo definirati ime izhoda in njegov podatkovni tip.

Za morebitne nastavitve grafičnega gradnika uporabljamo razred *Setting*. Razred nam omogoča shranjevanje nastavljene vrednosti. Za nastavitev ustvarimo spremenljivko, ki kaže na razred *Setting*, kateremu ob inicializaciji podamo privzeto vrednost nastavitve.

Za predstavitev grafičnih elementov na grafičnem gradniku vsebuje Orange svoje razrede, ki so vsebovani v paketu *widgets.gui*. S pomočjo teh razredov lahko implementiramo kontrole na grafičnem elementu.

[6]http://scikit-learn.org/

Razredi, s katerimi implementiramo kontrole prejemajo različne argumente, nekateri pa so skupni vsem, od teh so najpomembnejši:

- Widget - grafični gradnik, na katerem se bo element izrisal,
- master - objekt, ki vsebuje atribut, katerega želimo kontrolirati (običajno je to sam grafični gradnik - $self$),
- value - ime atributa, ki ga želimo kontrolirati,
- callback - funkcija, oziroma seznam funkcij, ki se kličejo ob spremembi vrednosti kontrole,

Poleg Orangevih elementov lahko seveda uporabljamo tudi osnovne PyQtjeve gradnike. V tej nalogi za prikaz posameznega okvirja uporabljamo PyQtjev gradnik $QPixMap$.

2.4 OpenCV

OpenCV je odprto-kodna knjižnica za slikovno in video analizo [1]. Knjižnica je napisana v programskem jeziku C, kar zagotavlja hitro in prenosljivo kodo, ponuja pa vmesnike do drugih programskih jezikov kot so Python, Java, C++ [9].

OpenCV ponuja učinkovite implementacije algoritmov s področja strojnega vida. Omogoča nam zajem, obdelavo in prikaz videa [9].

V jeziku Python OpenCV za shranjevanje slik uporablja Numpyjev podatkovni tip $array$, zato je mogoče za obdelavo uporabljati tudi Numpy funkcije.

V tej nalogi smo knjižnico uporabili za zajem (branje) videa, ter za izvedbo nekaterih operacij procesiranja, na primer za izgradnjo in ravnanje histograma slike, pretvorbo med barvnimi prostori, Gaussov filter in upragovanje slike.

2.5 PyQt4

PyQt je vezava priljubljenega orodja za razvoj grafičnih vmesnikov Qt za Python. Orodje Qt je implementirano v programskem jeziku C++. Sestavljeno je iz grafičnih gradnikov, s katerimi gradimo uporabniški vmesnik.

Uporaba PyQtja nam omogoča hiter razvoj grafičnih vmesnikov, ki so zelo prenosljivi, saj je koda programov enaka na vseh podprtih platformah [8].

Grafični vmesnik Orangea uporablja PyQt4.

Poglavje 3

Zgradba

V okviru naloge smo razvili 13 grafičnih gradnikov. Osem jih služi sami obdelavi, po en pa za branje in predvajanje videa, eden za ločevanje in dva za združevanje barvnih kanalov. Grafični gradniki za prenos podatkov uporabljajo razred *FrameData*, ki vsebuje določen okvir videa, ter nekatere uporabne podatke o videu.

Razvite gradnike lahko v grobem razdelimo v dve kategoriji: tiste, ki dedujejo razred *OWVideoWidget* in predstavljalo glavnino razvitih grafičnih gradnikov, ter ostale, bolj specializirane grafične gradnike, kot so na primer grafični gradnik za branje videa in gradnik za predvajanje videa, ter grafični gradniki za delo s posameznimi barvnimi kanali.

3.1 Razred FrameData

Razred *FrameData* predstavlja osnovni podatkovni tip, uporabljen v razvitih gradnikih. Uporabljamo ga za shranjevanje posameznega okvirja videa, ter podatkov o le-tem. Vsebuje:

- *frame* - okvir videa (Numpy polje),
- *fps* - število okvirjev na sekundo v videu,
- *length* - dolžina videa - število okvirjev v videu,

- $frame\_number$ - zaporedna številka okvirja.

Razred je možno v prihodnosti razširiti še z drugimi podatki.

Za inicializacijo razreda potrebujemo samo podatek $frame$, sicer lahko konstruktorju podamo še posamezne parametre ali pa že obstoječ primerek razreda $FrameData$. Če za inicializacijo uporabljamo obstoječ primerek, se pri inicializaciji vsi njegovi podatki razen okvirja kopirajo v nov primerek. Z uporabo takšnega načina inicializacije v naših grafičnih gradnikih lahko zagotovimo, da nam ob morebitnih razširitvah razreda $FrameData$ ne bo potrebno spreminjati veliko kode.

3.2 Razred OWVideoWidget

Razred $OWVideoWidget$ je osnova za razvoj grafičnih gradnikov za obdelavo videa znotraj orodja Orange. Razred deduje osnovi grafični gradnik orodja Orange $OWWidget$ in definira po en vhod in izhod - oba sta podatkovnega tipa $FrameData$. Definira tudi spremenljivko *cache* za shranjevanje najaktualnejšega prejetega okvirja, ter spremenljivko *thread*, kamor shranimo nit, v kateri se izvaja sama obdelava videa.

OWVideoWidget implementira metode, ki prejmejo $FrameData$, ki se pojavi na vhodu in ga pripravijo za obdelavo. Ko se na vhodu pojavi nov $FrameData$, se najprej kliče metoda *recieve*, ki prejet podatek shrani v spremenljivko *cache*, za tem pokliče metodo *start_process*. Metoda *start_process* preveri ali že obstaja nit za obdelavo. Če niti še ni, jo ustvari, ter v njej požene metodo *process_in_background*. Obdelava videa mora teči v svoji niti, sicer se grafični vmesnik Orangea blokira.

Metoda *process_in_background* pripravi prejet podatek na obdelavo. Iz spremenljivke *cache* vzame $FrameData$ in preveri prisotnost barvnih kanalov v vsebovanem okvirju. Preverjanje prisotnosti barvnega kanala izvedemo s seštevanjem elementov posameznega kanala, če je vsota manjša od 1, pomeni da barvni kanal v okvirju ni prisoten. Če kanal ni prisoten, pripadajočo spremenljivko postavimo na 0, sicer ostane 1.

Po preverjanju prisotnosti barvnih kanalov lahko začnemo s samo obdelavo okvirja. To izvedemo s pomočjo abstraktne metode *process_frame*, kateri pošljemo okvir, vrne pa obdelan okvir.

Grafični gradniki, ki dedujejo razred *OWVideoWidget* morajo predefinirati metodo *process_frame*, vanjo spravijo implementacijo operacije, ki jo naj bi izvajali nad okvirjem.

Po vrnitvi iz metode *process_frame* imamo obdelan okvir. V obdelanem okvirju glede na začetno prisotnost barvnih kanalov tiste, ki niso prisotni, pobrišemo. S tem se izognemo morebitnim motnjam, ki lahko nastanejo med obdelavo okvirja.

Brisanje kanala izvedemo z množenjem kanala s prej nastavljeno spremenljivko.

Po brisanju sestavimo nov *FrameData*, konstruktorju pošljemo stari *FrameData* in novi okvir. Nov *FrameData* pošljemo s pomočjo metode *send*. Metodi moramo podati ime izhoda in podatek, ki ga pošiljamo.

Poglavje 4

Predstavitev razvitih grafičnih gradnikov

V tem poglavju sta predstavljena delovanje in uporaba trinajstih grafičnih gradnikov, razvitih v okviru te naloge. Za potrebe testiranja smo uporabili prosto dostopni video Elephant's Dream[1]

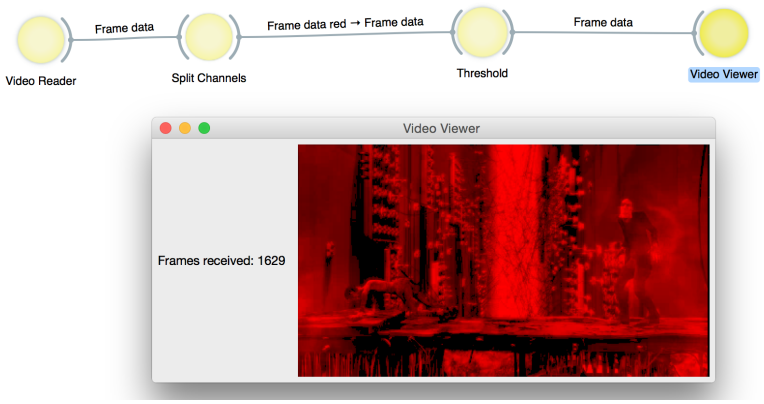
Razvili smo naslednje grafične gradnike:

- Video Reader - bralnik videa,
- Video Viewer - prikazovalnik videa,
- Video Delay - časovni zamik,
- Mirror - zrcaljenje,
- Brightness & Contrast - nastavljanje svetlosti in kontrasta,
- Noise - šum,
- Blur - glajenje,
- Sharpen - ostrenje,
- Histogram Equalization - razteg histograma,

[1] https://orange.blender.org

- Split Channels - ločevanje barvnih kanalov,
- Join Channels - združevanje barvnih kanalov,
- Join Channels - Synchronised - sinhrono združevanje barvnih kanalov.

Slika 4.1 prikazuje primer sheme za obdelavo videa znotraj Orangea. Video preberemo z grafičnim gradnikom Video Reader, nato z gradnikom Split Channels izločimo rdeč kanal, nad izločenim kanalom s pomočjo grafičnega gradnika Threshold izvajamo upragovanje, na koncu prikažemo video z gradnikom Video Viewer.

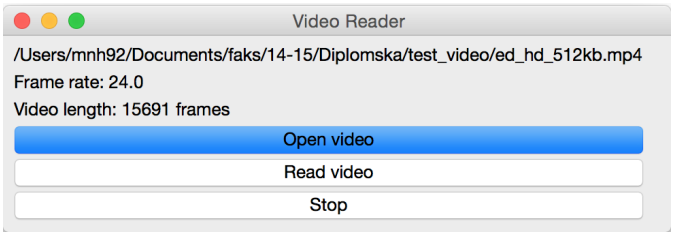


Slika 4.1: Primer sheme za obdelavo videa znotraj Orangea

4.1 Video Reader

Bralnik videa je namenjen branju videa iz datoteke in pošiljanju prebranih podatkov v nadaljnje procesiranje. V prihodnosti bi bilo možno gradnik razširiti z možnostjo zajema video posnetkov iz kamere.

Grafični gradnik je prikazan na sliki 4.2.



Slika 4.2: Grafični gradnik Video Reader

Kontrole zajemajo:

- Open video - odpre okno za izbiro video datoteke, katero želimo prebrati,
- Read videa - prične z branjem,
- Stop - prikaže se po začetku branja in omogoča njegovo ustavitev.

Poleg tega pa imamo na grafičnem gradniku še izpis naslednjih podatkov o samem videu:

- celotna pot do izbrane datoteke,
- fps,
- dolžina videa (število okvirjev).

Branje videa je izvedeno s pomočjo knjižnice OpenCV. Uporabimo objekt VideoCapture, konstruktorju podamo celotno pot do video datoteke. Iz tega objekta lahko s klicem *get*() dobimo podatke o videu, na primer dolžino videa in zaporedno številko okvirja, primer je prikazan v kodi 4.1.

Koda 4.1: Pridobivanje podatkov o videu - dolžina videa

```
self.length = int(vidcap.get(cv2.CAP_PROP_FRAME_COUNT))
```

Nad objektom nato kličemo metodo *read*. Metoda nam vrne boolean vrednost, ki predstavlja uspeh oz. neuspeh branja, ter prebran okvir kot

NumpPy *ndarray*. Prebrani okvirji so v formatu BGR, za nadaljnje delo večinoma uporabljamo RGB, zato se že na tem mestu izvede pretvorba s pomočjo OpenCVjeve funkcije *cvtColor*.

Glavna zanka za branje videa je prikazana v kodi 4.2. Nad videom kličemo metodo *read*, metoda nam vrne uspeh branja (*True* ali *False*) in prebran okvir. Dokler je branje uspešno in branja nismo ustavili, najprej s funkcijo *cvtColor* spremenimo barvni prostor prebranega okvirja iz BGR v RGB. Nato povečamo števec prebranih okvirjev. Okvir nato zapakiramo v razred *FrameData*, kamor shranimo še podatke o okvirju. *FrameData* pošljemo z metodo *send*. Za tem čakamo $\frac{1}{fps}$ sekund, s čimer poskrbimo da se bo video predvajal s (približno) enako hitrostjo kot izvirnik. Na koncu izvedemo še branje naslednjega okvirja.

Koda 4.2: Glavna zanka za branje videa

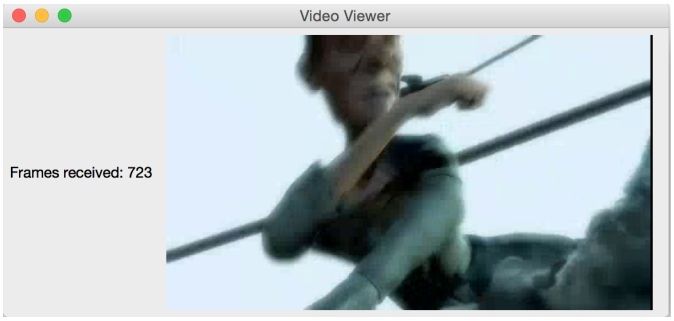
```
success, frame = vidcap.read()
while success and not self.stop:
    cv2.cvtColor(frame, cv2.COLOR_BGR2RGB, frame)
    self.frame_number += 1
    frame_data = FrameData(frame, self.fps, self.length,
        self.frame_number)
    self.send("Frame data", frame_data)
    time.sleep(1/self.fps)
    success, frame = vidcap.read()
```

Branje videa izvajamo v svoji niti, sicer se grafični vmesnik Orangea blokira dokler branje ni končano.

4.2 Video Viewer

Prikazovalnik videa je namenjen prikazu prebranih okvirjev. Grafični gradnik dobi na vhod polje *frame_data*. Polje vsebuje okvir videa, ter ostale podatke o videu. Prikaže trenuten okvir videa in število predvajanih okvirjev.

Gradnik je prikazan na sliki 4.3.



Slika 4.3: Grafični gradnik Video Viewer

Na glavno polje grafičnega gradnika postavimo oznako *image* znotraj katere bomo izrisovali slike. Metoda $receive(frame\_data)$ se kliče ob vsakem prejetem vhodu $frame\_data$. Za prikaz slike uporabimo razreda $QImage$ in $QPixMap$ iz paketa $PyQt4$. Za vsak prejet vhod iz $FrameData$ poberemo okvir in iz katerega ustvarimo $QPixMap$, ki ga prikažemo znotraj oznake *image*. Poleg tega še štejemo število prejetih okvirjev.

4.3 Video Delay

Gradnik je namenjen simulaciji časovnega zamika do katerega pride pri obdelavi videa. Uporaben je za testiranje.

Grafični gradnik deduje razred $OWVideoWidget$. Zamik je implementiran v metodi $process\_frame(frame)$ in ne obsega drugega kot klic metode $sleep(time)$, kjer je $time$ čas spanja v sekundah.

Grafični gradnik ima kontrolo Delay s katero izberemo zamik. Vrednost je v sekundah od 0 do 10.

4.4 Mirror

Gradnik je namenjen navpičnemu in vodoravnemu zrcaljenju videa. Vsebuje kontrole za izbiro osi zrcaljenja (x, y ali obe).

Gradnik deduje razred $OWVideoWidget$, v metodi $process\_frame$ glede na izbrani kontroli zrcalimo preko osi x (koda 4.3), oziroma preko osi y (koda 4.4).

Koda 4.3: Zrcaljenje preko x osi

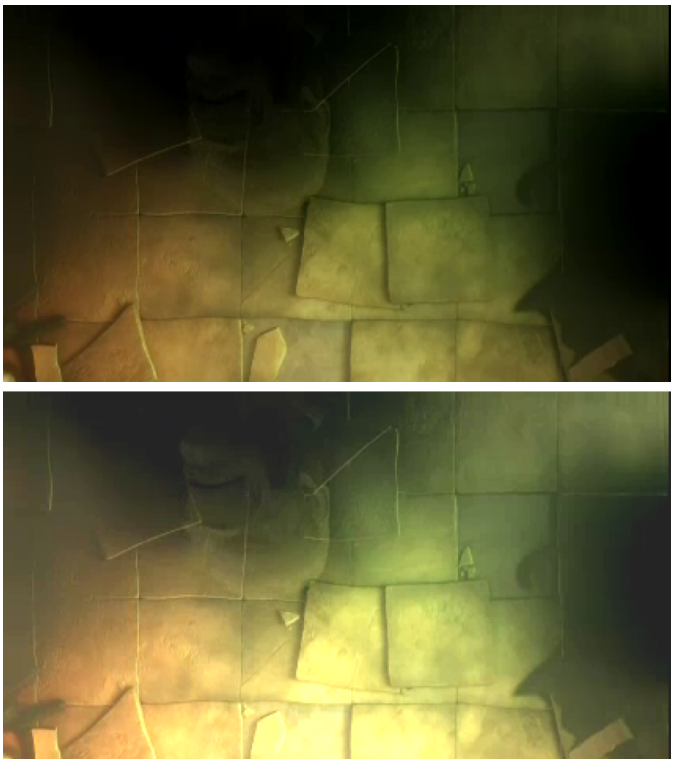
```
frame_out = np.copy(frame_out[:, ::-1, ...])
```

Koda 4.4: Zrcaljenje preko y osi

```
frame_out = np.copy(frame_out[::-1, ...])
```

4.5 Brightness & Contrast

Grafični gradnik omogoča nastavljanje svetlosti in kontrasta. Primer uporabe je prikazan na sliki 4.4. Zgornja slika je original, spodnja pa ima svetlost povečano za 25, ter vrednost kontrasta 1.50.



Slika 4.4: Primer uporabe grafičnega gradnika Brightness & Contrast

Svetlost pomeni intenziteto barve v posameznem pikslu. Spremembo svetlosti dosežemo s prištevanjem, oz. odštevanjem konstante od vrednosti pikslov.

Kontrast je definiran kot lokalna sprememba svetlosti [7]. Kontrast spreminjamo z množenjem s skalarjem.

Grafični gradnik deduje razred $OWVideoWidget$. Implementacija spreminjanja svetlosti in kontrasta je zajeta v metodi $process\_frame$.

S pomočjo Numpyjeve metode *clip* poskrbimo, da pri prištevanju svetlosti ne zaidemo izven intervala $[0, 255]$, nato izvedemo seštevanje. Spremembo kontrasta izvedemo z OpenCV metodo $multiply()$, ki sama poskrbi, da ne zaidemo izven intervala $[0, 255]$.

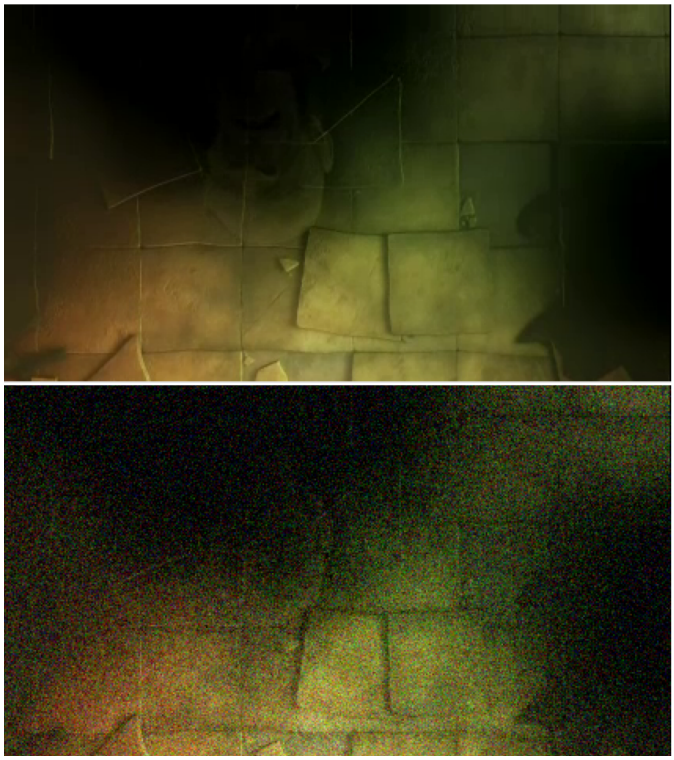
4.6 Noise

Grafični gradnik omogoča vnašanje šuma v sliko. Uporabljamo Gaussov šum, ki je oblika šuma, kjer šum ni odvisen od signala. Vrednosti, ki jih šum generira, so porazdeljene z Gaussovo funkcijo 4.1.

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{4.1}$$

kjer je μ povprečna vrednost in σ standardni odklon [7].

Na sliki 4.5 je prikazana uporaba grafičnega gradnika Noise, generiran je Gaussov šum s standardno deviacijo 25 in povprečno vrednostjo 0.



Slika 4.5: Primer uporabe grafičnega gradnika Noise

V grafičnem gradniku generiramo šum s povprečno vrednostjo 0, standardni odklon pa nastavimo s kontrolo, izberemo lahko vrednost med 1 in

25. Napako v sliko vnesemo s prištevanjem generiranega šuma k sliki.

Grafični gradnik deduje razred $OWVideoWidget$, implementacija je zajeta v metodi $process\_frame(frame)$.

Šum generiramo z NumPy funkcijo *normal*, kjer kot parametre podamo povprečno vrednost, standardni odklon in dimenzije okvirja. Za tem šum prištejemo okvirju z OpenCVjevo metodo *add*. Uporaba metode *normal* je prikazana v kodi 4.5

Koda 4.5: Generiranje Gaussovega šuma

```
noise = np.random.normal(0, self.std_dev, (h, w, ch))
```

4.7 Blur

Grafični gradnik se uporablja za glajenje slik. Uporabljamo glajenje z Gaussovim jedrom.

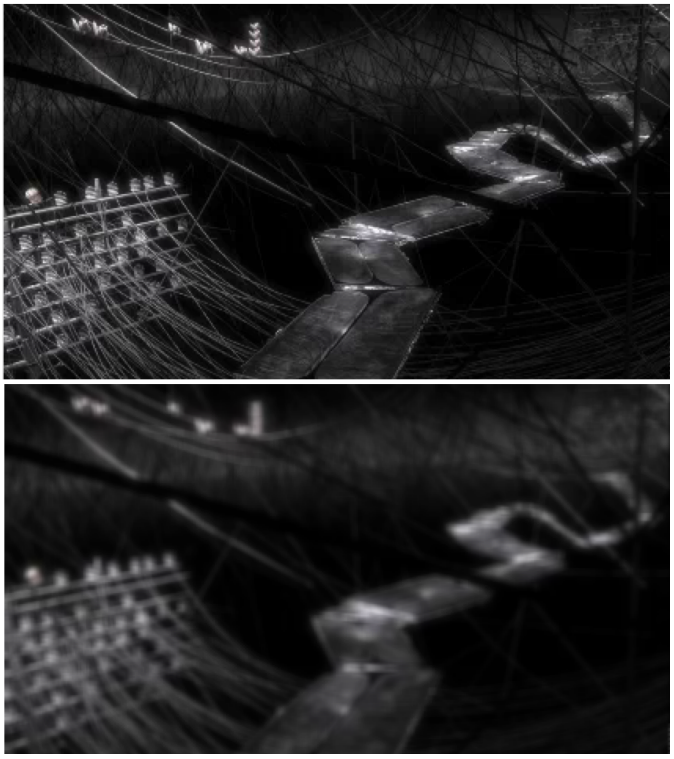
Glajenje slik uporabljamo za zmanjšanje v sliki prisotnega šuma [7].

Glajenje z Gaussovim jedrom je konvolucija slike z Gaussovim filtrom, formula je prikazana v enačbi 4.2, kjer sta x in y koordinati piksla slike, σ pa je standardni odklon.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{4.2}$$

V našem grafičnem gradniku uporabljamo jedro velikosti 5×5.

Grafični gradnik vsebuje kontrolo za nastavljanje standardnega odklona Gaussove funkcije s katero gladimo. Izberemo lahko vrednost med 0 in 25. Primer uporabe je prikazan na sliki 4.6, standardni odklon je 25.



Slika 4.6: Primer uporabe grafičnega gradnika Blur

Grafični gradnik deduje razred $OWVideoWidget$. Glajenje je implementirano v metodi $process\_frame(frame)$, izvedemo ga z OpenCVjevem razredom $GaussianBlur()$, kateremu podamo izvorno sliko (okvir videa), velikost jedra, ter standardni odklon.

Uporaba razreda $GaussianBlur$ je prikazana v kodi 4.6.

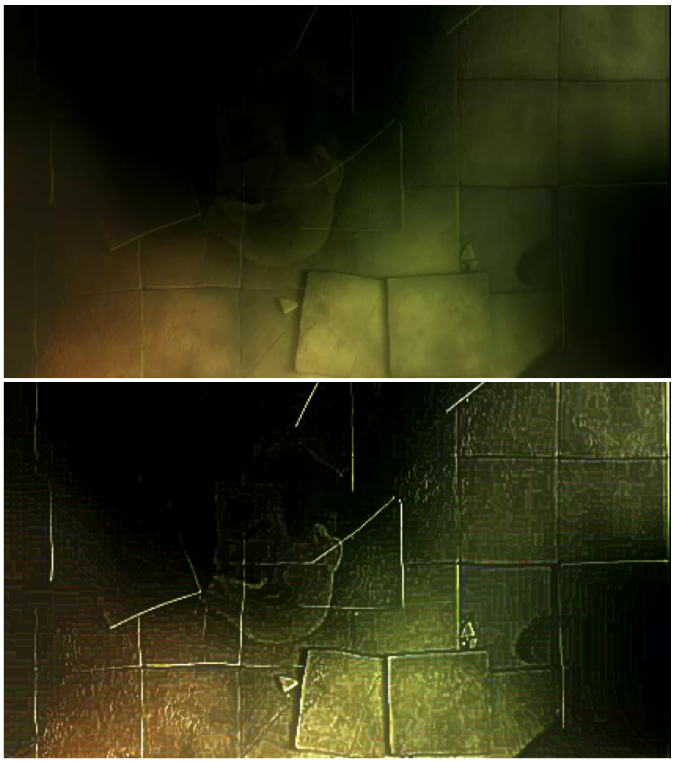
Koda 4.6: Uporaba razredga Gaussian Blur

```
frame = cv2.GaussianBlur(frame, (5, 5), self.sigma)
```

4.8 Sharpen

Grafični gradnik je namenjen ostrenju slike. Izvajamo ostrenje z glajenjem. Slika 4.7 prikazuje primer uporabe. Gradnik vsebuje kontrolo za izbiro uteži

za glajenje, v primeru je uporabljena utež 11.



Slika 4.7: Primer uporabe grafičnega gradnika Sharpen

Pri procesu ostrenja z glajenjem močno glajeno sliko odštejemo od originalne. Nato se slika rekonstruira kot utežena vsota originalne slike in maske.

Grafični gradnik deduje razred $OWVideoWidget$, glajenje je zajeto v metodi $process\_frame$. Filter dobimo s pomočjo OpenCV razreda $GaussianBlur$, kjer uporabimo velikost jedra $(3, 3)$, ter standardni odklon 0 (koda 4.7).

Koda 4.7: Filter za glajenje z ostrenjem

```
filter = cv2.GaussianBlur(frame, (3, 3), 0)
```

Za tem z odštevanjem dobimo filtrirano sliko (koda 4.8)

Koda 4.8: Filtriranje originalne slike

```
filtered = cv2.subtract(frame, filter)
```

Ostreno sliko dobimo kot utežen seštevek filtrirane slike in originalne slike, kjer je parameter *alpha* utež, ki vpliva na moč ostrenja. Proces ostrenja je prikazan v kodi 4.9.

Koda 4.9: Ostrenje z glajenjem

```
sharpened = cv2.addWeighted(frame, 1, filtered, self.alpha, 0)
```

4.9 Histogram Equalization

Grafični gradnik izvede ravnanje histograma. Histogram sivinskih nivojev slike nam pokaže frekvenco pojavitev posameznih sivinskih nivojev v sliki [7]. Z ravnanjem histograma dosežemo enakomerno porazdelitev sivinskih nivojev v sliki skozi celoten spekter, torej je rezultat ravnanja slika z izboljšanim kontrastom [7]. Primer uporabe je prikazan na sliki 4.8, zgornja slika je original, nad spodnjo pa smo izvedli ravnanje histograma.

Slika 4.8: Primer uporabe grafičnega gradnika Histogram Equalization

Ravnanje histograma se običajno izvaja nad sivinsko sliko, lahko pa sliko pretvorimo v barvni prostor HSV, kjer je posamezen piksel predstavljen z vrednostmi odtenka, zasičenosti in svetlosti. Ravnanje histograma nato izvedemo nad komponento svetlosti.

Pretvorbo med barvnimi prostori izvajamo s pomočjo OpenCVjeve funkcije *cvtColor*(). Koda 4.10 prikazuje uporabo funkcije *cvtColor*, funkciji podamo izvorno sliko in kodo za želeno pretvorbo, vrne nam sliko, pretvorjeno v želen barvni prostor.

Koda 4.10: Uporaba funkcije cvtColor

```
frame_hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)
```

Ko imamo sliko v HSV barvnem prostoru, izločimo njeno komponento svetlosti (V), ter nad njo izvedemo ravnanje histograma s pomočjo OpenCV-

jeve funkcije *equalizeHist*. Uporaba funkcije *equalizeHist* je prikazana v kodi 4.11. Funkciji podamo izvorno sliko z enim kanalom, vrne pa nam sliko, nad katero je bilo izvedeno ravnanje histograma.

Koda 4.11: Uporaba funkcije equalizeHist

```
frame_v = cv2.equalizeHist(frame_v)
```

Ravnano komponento svetlosti nato vključimo nazaj v izvorno sliko in jo pretvorimo nazaj v barvni prostor RGB.

4.10 Threshold

Grafični gradnik izvede upragovanje slike. Upragujemo lahko sivinsko sliko ali po posameznem barvnem kanalu slike.

Če upragujemo sivinsko sliko imamo na voljo dve vrsti upragovanja: binary, ki vrednost pikslov, večjih ali enakih pragu postavi na 255, ostale pa na 0, ter binary inverted, ki naredi obratno. Primer uporabe upragovanja sivinske slike je prikazan na sliki 4.9, zgornja slika je original, nad spodnjo pa smo izvedli upragovanje po vrednosti 234, uporabljamo binarno upragovanje.

Slika 4.9: Primer uporabe grafičnega gradnika Threshold - upragovanje sivinske slike

Če upragujemo po posameznem barvnem kanalu najprej s kontrolo izberemo ustrezen kanal. Upragovanje poteka tako, da se piksli z vrednostmi pod pragom postavijo na 0, ostali pa so nespremenjeni. Primer upragovanja po posameznem kanalu je prikazan na sliki 4.10. Upragujemo po rdečem kanalu, zgornja slika je original, na spodnji pa upragujemo po vrednosti 232.

Slika 4.10: Primer uporabe grafičnega gradnika Threshold - upragovanje po posameznem kanalu

Ker upragovanje poteka nad sivinsko sliko moramo najprej narediti ustrezno pretvorbo s pomočjo funkcije *cvtColor*, kateri podamo kodo pretvorbe *COLOR_RGB2GRAY*.

Samo upragovanje naredimo s pomočjo OpenCVjeve funkcije *threshold*. Funkcija izvede upragovanje po želeni vrednosti nad sliko z enim kanalom.

Uporaba funkcije *threshold* je prikazana v kodi 4.12, funkciji podamo sliko z enim kanalom, vrednost, ki predstavlja prag, maksimalno vrednost v sliki, ter vrsto upragovanja, ki ga želimo izvesti.

Koda 4.12: Uporaba funkcije threshold

```
ret, frame = cv2.threshold(frame_gray, self.value, 255,
    cv2.THRESH_BINARY)
```

4.11 Split Channels

Grafični gradnik razdeli sprejet okvir po barvnih kanalih. To nam omogoča neodvisno obdelavo posameznih barvnih kanalov videa. Primer uporabe je prikazan na sliki 4.11, z grafičnim gradnikom Split Channels smo izločili zelen barvni kanal, zgornja slika prikazuje original, spodnja pa izločen barvni kanal.

Slika 4.11: Primer uporabe grafičnega gradnika Split Channels - prikazan le zelen kanal

Grafični gradnik Split Channels je v implementaciji podoben razredu $OWVideoWidget$, le da imamo tu tri izhode - po enega za vsak barvni kanal.

Razbijanje na posamezne barvne kanale teče, tako kot vse ostale obdelave, v svoji niti.

V niti se za vsak prejet okvir ustvarijo tri kopije (ena za vsak barvni kanal), nato se vrednosti kanalov, ki ne bosta prisotna postavijo na 0. S tem

vseeno ohranimo dimenzije originalnega okvirja, tako da ga lahko enostavno uporabljamo v ostalih grafičnih gradnikih.

Izločanje rdečega barvnega kanala je prikazano v kodi 4.13. Najprej z metodo *copy* ustvarimo kopijo originalnega okvirja. Nato zelen in moder kanal postavimo na 0. Na koncu ustvarimo nov primerek razreda $FrameData$ na podlagi prejetega primerka.

Koda 4.13: Izločanje barvnega kanala

```
frame_red = np.copy(cur_fd.frame)
frame_red[:, :, 1] = 0
frame_red[:, :, 2] = 0
frame_data_red = FrameData(frame_red, cur_fd)
```

4.12 Join Channels

Grafični gradnik združi prejete barvne kanale brez uporabe sinhronizacije. Ima tri vhode, po enega za vsak kanal. Za združevanje potrebuje povezan vsaj en vhod. Vhode lahko poljubno zamenjujemo, prav tako ni nujno, da so vsi del istega videa, pomembno je le ujemanje dimenzij okvirja.

Ker grafični gradnik ne implementira sinhronizacije okvirjev, lahko v združenem videu pride do motenj.

Ker imamo tri vhode, ima grafični gradnik tri metode, ki se prožijo ob prejetju podatka na vhodu: $receive\_x$, kjer x predstavlja barvni kanal.

Metode prejet vhod shranijo v ustrezno mesto v polju *cur*, ter s pomočjo metode *add_data* shranijo podatke o okvirju.

Metoda *_process_in_background* teče v svoji niti. Najprej preverimo ali imamo na voljo kakšen okvir, če ga imamo okvirje združimo s pomočjo metode $join\_frames$, nato pa pošljemo z metodo *send*

V metodi $join\_frames$ ustvarimo prazno polje enakih dimenzij kot vhod, nato enostavno prištejemo posamezne kanale, tako da ni potrebno preverjanje kateri kanali so na vhodu.

Motnje v video se pojavljajo, ker združevanje izvedemo takoj, ko imamo na voljo okvir. Torej lahko pošljemo samo en kanal ali združimo okvirje iz različnih delov videa. Prednost tega je, da lahko združujemo tudi okvirje, ki prihajajo iz različnih videov. Poleg tega obdelava poteka hitreje, saj je najmanjša hitrost združevanja enaka hitrosti prejema najhitrejšega kanala.

4.13 Join Channels - Synchronised

Grafični gradnik združi barvne kanale z uporabo sinhronizacije. Barvne kanale združimo šele, ko najdemo trojico, katerih razlika zaporednih številk okvirjev je znotraj izbrane meje. Tako je hitrost združevanja odvisna od hitrosti prejemanja najpočasnejšega kanala.

Imamo tudi možnost omejitve časa čakanja na prejem na najpočasnejšem kanalu. Če uporabljamo omejitev čakanja, se prejemanje na hitrejših dveh kanalih ustavi, dokler ne uspemo sestaviti celotne trojice.

Uporaba sinhronizacije močno upočasni združevanje, a se lahko izognemo morebitnim motnjam v videu.

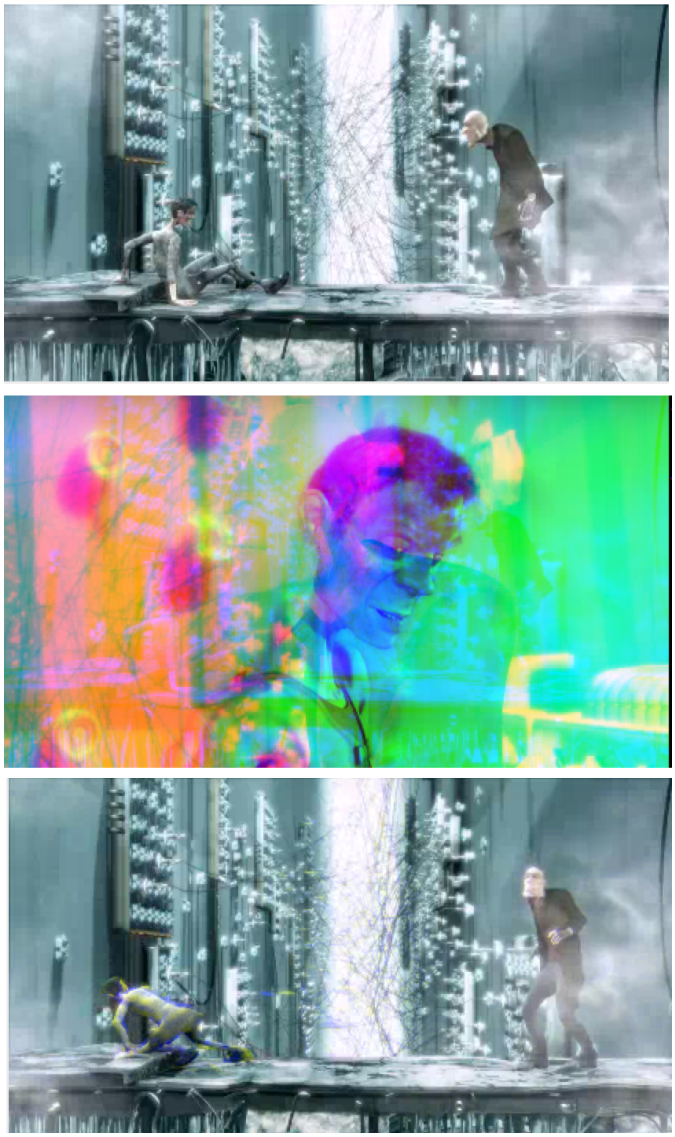
Glavna težava sinhronizacije se pojavi, ker ne moremo zagotavljati, da bodo v grafični gradnik vedno prispeli vsi trije deli določenega okvirja (trije deli z isto zaporedno številko okvirja), saj se lahko kateri izgubi med obdelavo. Zaradi tega čakanje na prejem vseh treh delov nekega okvirja ni smiselno.

Za vsak kanal uporabljamo medpomnilnik v katerem hranimo prejete dele. Predpostavljamo, da okvirji na določenem kanalu prihajajo v istem zaporedju, kakršno je v izvornem videu.

Najprej ugotovimo na katerem kanalu prejemamo najpočasneje, nato pa za vsak prejet del okvirja na tem kanalu v ostalih dveh medpomnilnikih poskusimo najti ujemanje. Možnost, da najdemo ujemanje lahko povečamo tako, da namesto točnega ujemanja zaporednih številk okvirja iščemo znotraj določenega intervala.

Slika 4.12 prikazuje uporabo grafičnega gradnika Join Channels - Synchronised. Video smo razbili na posamezne barvne kanale, nato pa s pomočjo

grafičnih gradnikov Delay za vsakega uvedli različen zamik pred združevanjem. Rdeč kanal ima zamik ene sekunde, zelen treh, moder petih sekund. Kanale smo nato združili z obema grafičnima gradnikoma za združevanje kanalov. Zgornja slika prikazuje originalen video, srednja kanale, združene brez sinhronizacije, spodnja pa kanale, združene z uporabo sinhronizacije, kjer je meja za združevanje nastavljena na 5.



Slika 4.12: Združevanje kanalov z različnimi zamiki brez in s sinhronizacijo

Za shranjevanje prejetih delov okvirja uporabljamo za vsak kanal po en medpomnilnik dolžine 256. Ob prejetju podatka se kliče metoda *receive_x*.

V metodi *receive_x* najprej preverimo ali je prejemanje na tem kanalu ustavljeno, če ni, preverimo čas, ki je pretekel od inicializacije gradnika, do prejetja podatka na tem kanalu. Nato s pomočjo metode *add_to_cache* shranimo podatek v ustrezen medpomnilnik, ter s pomočjo metode *add_data* shranimo še ostale podatke o videu. Metoda *add_data* je enaka tisti, ki jo uporablja grafični gradnik Join Channels.

Metodi *add_to_cache* pošljemo okvir in medpomnilnik, v katerega želimo shraniti okvir. Najprej preverimo ali je medpomnilnik poln, če je iz njega odstranimo najstarejši okvir. Za tem shranimo nov okvir.

Če imamo v vseh treh medpomnilnikih na voljo vsaj en okvir, v ločeni niti izvajamo metodo *synchronise*. V tej metodi najprej ugotovimo, na katerem kanalu prejemamo najpočasneje. Ugotavljanje prikazuje koda 4.14. Čase prejemanja, ki jih nastavimo v metodah *receive_x* shranimo v polje, nato nad poljem kličemo metodo *max*, ki vrne največjo vrednost v polju. To vrednost podamo metodi *index*, ki jo kličemo nad poljem in ki nam vrne mesto iskane vrednosti v polju.

Koda 4.14: Ugotavljanje najpočasnejšega kanal

```
times = self.time_r, self.time_g, self.time_b
self.slowest = times.index(max(times))
```

Ko imamo najpočasnejši kanal, ustvarimo prazno polje, kamor bomo shranjevali dele okvirja. Vanj shranimo zadnji prejet okvir najpočasnejšega kanala, ter shranimo njegovo zaporedno številko. Nato poskusimo najti ujemanje v ostalih dveh kanalih. Ujemanje iščemo z metodo $find\_frame$.

Metodi $find\_frame$ pošljemo zaporedno številko okvirja in medpomnilnik, v katerem iščemo. Metoda v izbranem medpomnilniku išče ujemanje z iskano številko okvirja. Metoda je prikazana v kodi 4.15.

Če najdemo ujemanje znotraj intervala, si zapomnimo okvir in njegovo mesto v medpomnilniku. Če najdemo točno ujemanje, vrnemo okvir in nje-

govo mesto v pomnilniku. Če smo zašli izven intervala, vrnemo najboljše ujemanje.

Koda 4.15: Metoda find_frame

```
cur_frame = None
cur_i = None
   for i, fd in enumerate(cache):
         if abs(fd.frame_number - frame_number) <= self.tolerance:
            cur_frame = fd.frame
               cur_i = i
         if fd.frame_number == frame_number:
            return fd.frame_number, i
         if fd.frame_number > (frame_number + self.tolerance):
               return cur_frame, cur_i
```

Če smo našli ustrezno trojico iz medpomnilnikov, pobrišemo najdeni okvir in vse starejše okvirje. Za tem dele združimo s pomočjo metode *join_frames*, ki je enaka tisti v grafičnem gradniku Join Channels. Združen okvir skupaj s podatki o videu zapakiramo v razred *Frame_Data* in pošljemo z metodo *join*.

Če trojice ne najdemo, najprej ugotovimo ali uporabljamo omejitev čakanja. Če omejitev uporabljamo, preverimo ali je kateri od kanalov ustavljen. Za vsak ustavljen kanal preverimo ali ga je najpočasnejši kanal prehitel, torej ali je zaporedna številka okvirja v medpomnilniku kanala manjša od zaporedne številke okvirja zadnjega prejetega okvirja na najpočasnejšem kanalu. Če smo ustavljen kanal prehiteli, spet sprostimo prejemanje na tem kanalu.

Če noben od kanalov ni ustavljen preverimo, koliko časa je minilo od zadnje poslane trojice. Če je čas večji od izbrane meje, ustavimo prejemanje.

Poglavje 5

Sklepne ugotovitve

V prvem delu smo predstavili tehnologije in orodja, ki smo jih uporabljali v okviru naloge. V naslednjem poglavju smo predstavili zgradbo razvitih grafičnih gradnikov. Sledil je še podroben pregled razvitih grafičnih gradnikov in implementacij različnih operacij obdelave videa.

Z razvitima razredoma *FrameData* in *OWVideoWidget* imamo dober temelj za hiter in enostaven razvoj novih gradnikov za obdelavo videa.

Razred *FrameData* je enostavno razširljiv, ne da bi morebitne razširitve vplivale na že razvite gradnike.

Razred *OWVideoWidget* nam omogoča enostaven razvoj novih grafičnih gradnikov z implementiranjem le ene metode.

Razviti gradniki dobro prikazujejo možnosti obdelave videa v orodju Orange. Zajeli smo dober nabor pogosto uporabljenih operacij obdelave videa. Razviti gradniki omogočajo enostavno izvajanje zaporedja različnih video obdelav.

Možnost za izboljšavo je predvsem v večjem izkoriščanju možnosti delitve videa po posameznih barvnih kanalih.

Kljub možnostim za izboljšavo ocenjujemo, da smo dosegli zastavljeni cilj.

Literatura

[1] Ivan Culjak, David Abram, Tomislav Pribanic, Hrvoje Dzapo, and Mario Cifrek. A brief introduction to opencv. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 1725–1730. IEEE, 2012.

[2] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, et al. Orange: data mining toolbox in python. *The Journal of Machine Learning Research*, 14(1):2349–2353, 2013.

[3] Janez Demšar and Blaž Zupan. Orange: Data mining fruitful and fun-a historical perspective. *Informatica*, 37(1), 2013.

[4] Hans Petter Langtangen. *A primer on scientific programming with Python*. Springer, 2009.

[5] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[6] Fernando Perez, Brian E Granger, and John D Hunter. Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2):13–21, 2011.

[7] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[8] Mark Summerfield. *Rapid GUI programming with Python and Qt: the definitive guide to PyQt programming*. Pearson Education, 2007.

[9] Brian Thorne. Introduction to computer vision in python. *University of Canterbury. New Zealand*, 2009.

[10] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.